Intel® Compilers

Problem Isolation: Tools and Techniques

Software Products Division

Software and Solutions Group

www.intel.com/software/products

www.intel.com/software/products

Agenda

- Introduction
- Compile-Time Errors
- Run-Time Errors
- Problem Report Submission

Problem Statement

- Fact: All software makes mistakes *occasionally*
- Production software is tested rigorously
  - But customers still find ways to break it
  - No one can test all possible inputs
- Higher levels of optimization from a compiler can evoke compiler errors or, in lots of cases, application errors

We need to pinpoint issues to a real bug in the compiler, or application error, and get the right info to the development team for timely fixes.

Submitting a Problem Report

- Need detailed description of the problem and how to reproduce it
  - Descriptive title includes application name (if appropriate)
  - Exact command-line used to invoke the compiler
  - Failure message(s)
  - Details, details, details
- Include Package ID – *icid* utility outputs version info
- Indicate “Customer Impact”
- Provide complete environment description
- **Small, IP-free test case - even for run-time errors**

The better the problem report, the faster the fix. Intel provides the tools and methods to produce excellent problem reports.

Submitting a Problem Report Examples

- Not Useful
  - I get a debugbreak error message when I use ecl with debug turned on. Sorry, I can't send you any code as it is proprietary.
- Better, but too big of a test case
  - ECL reports –debugbreak error when using ASSERT in debug mode. I tried to attach my MSVC project, but I got errors. The file size is 32MB. I'll try to upload the file again.

intel

Submitting a Problem Report Examples

- Good Problem Report

Error: identifier "__debugbreak" is undefined

Compile the following code with

ecl -c /D _DEBUG bugtest.cpp

```
#include <afxext.h>          // MFC extensions
int main(int argc, char* argv[])
{
  printf("Hello World!\n");
  return 0;
}
```

```
> Intel(R) C++ Compiler for Itanium(TM)-based applications
> Version 5.0.1, Build 20010504
> Copyright (C) 1985-2001 Intel Corporation.  All rights reserved.
>
> ecl: NOTE: The evaluation period for this product ends on
> 31-jul-2001 UTC.
> bugtest.cpp
> H:\\apps\\mssdk\\Include\\prerelease\\Win64\\MFC\\afxtls_.h(60
> ): error: identifi
> er "__debugbreak" is undefined
>         { ASSERT(m_pHead == NULL); m_nNextOffset = nNextOffset; }
```

Two Kinds of Compiler Problems

- Problems during compilation or linking
  - Compiler faults or internal errors
  - Link-time errors
  - *Not* syntax or semantic errors
    - Those are *your* problem…
    - Find your local language standards expert
- Problems during runtime
  - Compilation seems to work, and link is OK
  - Application fails regression tests, or
  - *Your* customer reports errors in *your* application
    - Need to determine where problem actually lies

This presentation will show you tools and techniques to help *both* situations.

Example Situations

- Compiler bails out on a large compilation unit, but code:
  - Is too big to be a reasonable test case (> 100 lines, eg)
  - Contains sensitive intellectual property
- App built **/Od** passes regression tests, built with **/O3** does not
- App built with compiler A works, but not when built with compiler B
- Customer discovers a new problem in the built app
  - Need to isolate the cause – could be an application error

A Word of Caution

- Commonly, porting apps to a new platform and new optimization levels points out application and test suite problems – not compiler bugs:
  - FP-precision differences: higher opts keep data in registers longer – data is truncated when writing to/from memory (80bit vs. 64bit)
    - Apps and test suites sometimes tuned for an exact match in compares – differences result
  - Data size differences – 32-bit vs. 64-bit
  - Non-standard language issues – makes ports harder since not all compilers support each other's nuances
  - High optimization can expose app errors also:

Can't just blame the compiler – make sure it is a valid issue.

The Compiler Problem Isolation Aids

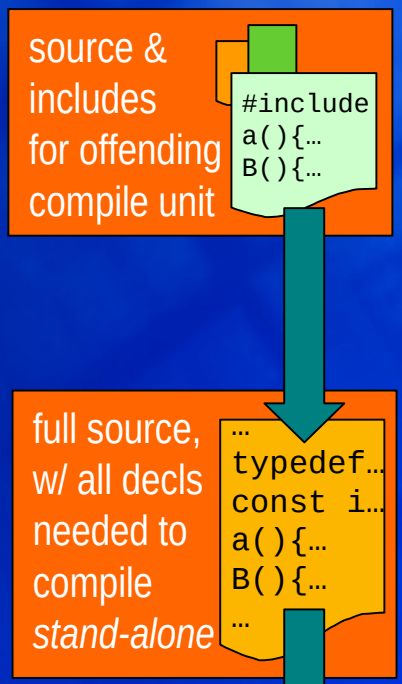
- Source code test case reducer - *icpi*
- IP protection for your code – *obfuscateCsource*

The better the problem report, the faster the fix. Intel provides the tools and methods to produce excellent problem reports.

Agenda

- Introduction
- Compile-Time Errors
- Run-Time Errors
- Problem Report Submission

Generate a *Portable* Test Case



- *First*, Produce dependency-free source sample
  - Use *all* options you'd normally compile with, *except*
  - Replace **/c** with **/E** to get pre-processor output

```
icl /E /D… /D… /I… … /O? … xxx.c > xxx.i
```

- *Second*, Compile to reproduce failure:
  - omit **/I** and **/D** options (they're done)
  - Use **/c** and other options affecting code generation

```
icl /c … /O? …  xxx.i
```

Free-standing code sample makes problem reproduction *portable*.

Maybe not *Yet* a Good Test Case…

- Might be too big
  - Preprocessed source can be 10s of thousands of lines
  - Smaller test cases quicker repairs
- Might reveal private intellectual property
  - May violate your company policy
  - You might *not* want to reveal what you're working on…
  - We'll cover this tool later

Tools to help *both* situations.

Reducing the Test Case Size

- **`icpi`** – Intel Compiler Problem Isolator
  - You specify command line (and error if needed)
  - **`icpi`** iteratively removes code sections and recompiles
  - Looks for error pattern matches, logs all work
  - Uses knowledge of compiler options, C/C++ syntax
  - Runs autonomously: overnight, over weekend
  - Installed with the compiler (all platforms)
  - icpi only supports C/C++ not Fortran

Let the computer try all the possible reproduction cases.

icpi Syntax

- **icpi [options] filename**
- Common option examples
  - **/compiler="icl –O3 –QxW"**
  - **/errormatch="fatal"**
  - **/logfile=<path>**
  - **/verbose**
  - **/help**
- Example:

```
icpi /compiler="icl –O3" /verbose xxx.i
```

What if `icpi` Doesn't Converge?

- BKMs for manual test case reduction:
  - First, reduce time per compilation
    - Start with "clean" preprocessed file
    - Determine "lowest" optimizations required (/Od is fastest)
  - Isolation suggestions
    - Start at the bottom of the file and work upward
    - Remove (or #ifdef / #endif) whole functions
    - Remove basic blocks in bad function (just leave {})
    - Remove unreferenced functions, classes and members
  - May need to iterate over file
  - Recompile and verify error *often*
    - Copy "bad" files to a save area

Protecting *Your* Intellectual Property

```
…
int scram(float temp, double
  …
  if ctrlRod(float position,
    …
    throw Emergency(…
    …
  }
  …
```

translated-idents.log

```
…
I0236 scram
I0
I0
I0
I0
I0
I0
I0
…
```

```
…
int I0236(float I0106, float
  …
  if I0034(float I0005,
    …
    throw I1069(…
    …
  }
  …
```

What if your source has features you'd rather an outside party *didn't* see?

- Start from pre-processed **`.i`** file (reproduction case or **`icpi`** output)
- Use **`obfuscateCsource`** to turn source identifiers into "license plate" numbers
  - Even translates "words" in *comments*
  - *Obfuscated* source *still* compilable
  - *Obfuscated* source sent in with problem report
- Only *you* have the name/identifier back-translation key
  - Only *you* can refer to original source to answer any questions

Added `obfuscateCsource` guidelines

- String literals *not* obfuscated
  - `Eg, "Control rod position %ld, …\n"` – Hmmm…
  - Check those yourself if OK to send out
  - Edit **`obfuscateCsource`** yourself
- Test-compile obfuscated source
  - Should compile just like original
    - Has worked for a huge variety of C and C++ source, but not yet tested with *yours*
  - Sorry: no FORTRAN obfuscation available
- Note: Available on request from Intel Premier Support

Agenda

- Introduction
- Compile-Time Errors
- Run-Time Errors
- Problem Report Submission

Why Not Just Use a Debugger?

- Using a debugger has issues
  - Not necessarily deterministic; time consuming
  - Many bunny trails before seeing actual problem
  - Often requires specialized, deep knowledge of app
- This presentation describes
  - Deterministic method
  - Requires knowledge of build environment, not app
  - Can estimate time to completion
  - Generates detailed info for problem reports
  - Works for a large class of problems
  - *Can be automated!*

Isolate problem first, *then* get into debugger.

Steps to Isolation: Pre-requisites

- Problem must be recognizable and repeatable
  - Best if a script can determine pass/fail
    - No human decision needed = automated
  - Smaller test case quicker repairs
- *Observable*, repeatable build mechanism
  - You can see/snapshot *all* the command-line options used to compile any **`.OBJ`** file
  - A command-line compile with the saved/observed options reproduces the problem

Armed with the above, you can do diagnostic *isolation* of compile-time problems.

Need a Valid Reference

- Given: Problem reproducible with icl optimized build
- Test: Does problem exist when built with icl /Od?
  - No:
    - Problem is likely in an icl optimization phase
    - icl /Od objects will be the reference
    - icl optimized objects will be the target
  - Yes: Test: Does problem exist when built with MSVC?
    - No
      - Problem is likely somewhere in icl
      - MSVC objects will be the reference
      - icl /Od objects will be the target
    - Yes
      - Problem is likely in your source – sorry!
      - Fire up your friendly debugger…
- Now we have a valid reference to compare against
- Note: Make sure your failure is not merely a FP precision issue!

Runtime Isolation Method

- Given: Valid reference objects and failing target objects
- Given: Ability to mix-and-match objects (ABI compatibility)
  - Can't have any special, trick assembly coding…
- Then: Perform mix-and-match builds and test
  - Search for smallest failing subset of objects

Then:

- Do a binary search to identify the failing object(s):
  - Compile half optimized, half un-optimized/debug, build, run
  - If it still fails, do the same to the optimized half until it works
  - If it works, optimize the unoptimized half until it fails
  - Repeat

How Much Have We Found Out?

- Now we know:
  - Command line and which compilation unit(s) evoke problem

Agenda

- Introduction
- Compile-Time Errors
- Run-Time Errors
- Problem Report Submission

Submit the Problem Report

- You now have a problem reproducer:
  - Stand-alone (already pre-processed) source
  - Name / number of object(s) showing the problem
  - Ability to generate *before* and *at* assembly
- Use obfuscation procedure discussed earlier if needed
  - Usually, source has some intellectual property in it
  - Repeat compile-to-*before* and compile-to-*at* steps with obfuscated source
    - **So you know what license-plate identifiers to mention in the problem report – function name, involved variables, etc.**

Report problem to https://premier.intel.com

intel

Summary

- You can protect your intellectual property when submitting a problem report
- Problem reports can be minimized
- Compiler tools help narrowing down the problem
- Different compilers can be used to check on each other and help isolation

This allows compiler engineers to quickly process problem reports, and add to their test cases.